6.4 Project Report

Vending Machine System

```
import java.util.*;

public class VendingMachine {  82 usages  zeelg
    private List<User> users;  8 usages
    private List<Product> inventory;  15 usages
    private Money currentSessionMoney;  13 usages
    private List<String> salesLog;  11 usages
```

```
     * @param operator the given operator performing the c
     */
    public void changePrice(Product item, double price, Op
        if (price < 0) {
            System.out.println("Invalid price. Price cann
            return;
        }
        if (operator.getAccessLevel() == AccessLevel.ADMIN
            item.setPrice(price);
            System.out.println("Succ                          + price);
            this.sale
        }
```

Submitted by:

Zeel Gajjar

Course:

420-SF2-RE

Data Structures and Objected-Oriented Programming

Table of Contents

DELIVERABLE 4

Project Description

The Vending Machine System is a Java-based application that simulates the functionality of a modern vending machine. It provides a robust and flexible framework for managing inventory, processing transactions, and handling user interactions, including buyers and operators with distinct roles and access levels. The system supports various product types, tracks sales, manages stock, and generates logs for operational and financial oversight.

Key Features:

- 🛒 **Inventory Management:** Tracks products (snacks, drinks), removes expired items, and supports restocking with capacity limits.
- 💰 **Transaction Processing:** Handles payments, calculates change, logs sales, and allows order cancellation.
- 🔄 **User Roles:** Buyers purchase items; operators (ADMIN/STAFF) manage stock and prices, with access control.
- ⚖️ **Product Sorting:** Sorts products by category or name using comparators.
- 📂 **Data Persistence:** Logs inventory and transactions to files; reads profit sheets for financial review.

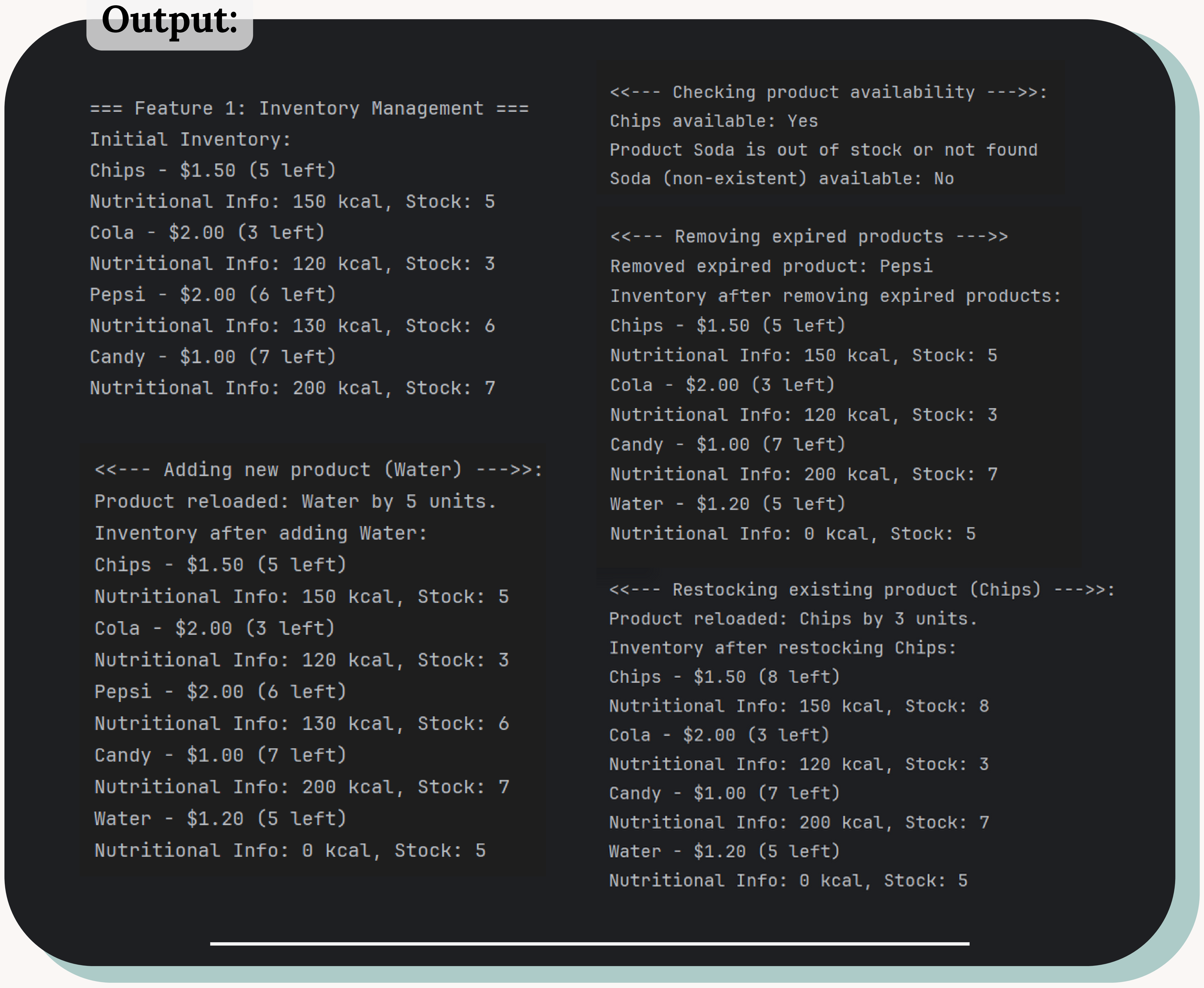
Program Features and Screenshots

(All execution code available in Main class)

1. Inventory Management:

This feature demonstrates the vending machine's ability to manage its product inventory.

Output:



```
=== Feature 1: Inventory Management ===
Initial Inventory:
Chips - $1.50 (5 left)
Nutritional Info: 150 kcal, Stock: 5
Cola - $2.00 (3 left)
Nutritional Info: 120 kcal, Stock: 3
Pepsi - $2.00 (6 left)
Nutritional Info: 130 kcal, Stock: 6
Candy - $1.00 (7 left)
Nutritional Info: 200 kcal, Stock: 7


<<--- Adding new product (Water) --->>:
Product reloaded: Water by 5 units.
Inventory after adding Water:
Chips - $1.50 (5 left)
Nutritional Info: 150 kcal, Stock: 5
Cola - $2.00 (3 left)
Nutritional Info: 120 kcal, Stock: 3
Pepsi - $2.00 (6 left)
Nutritional Info: 130 kcal, Stock: 6
Candy - $1.00 (7 left)
Nutritional Info: 200 kcal, Stock: 7
Water - $1.20 (5 left)
Nutritional Info: 0 kcal, Stock: 5
```

```
<<--- Checking product availability --->>:
Chips available: Yes
Product Soda is out of stock or not found
Soda (non-existent) available: No

<<--- Removing expired products --->>
Removed expired product: Pepsi
Inventory after removing expired products:
Chips - $1.50 (5 left)
Nutritional Info: 150 kcal, Stock: 5
Cola - $2.00 (3 left)
Nutritional Info: 120 kcal, Stock: 3
Candy - $1.00 (7 left)
Nutritional Info: 200 kcal, Stock: 7
Water - $1.20 (5 left)
Nutritional Info: 0 kcal, Stock: 5

<<--- Restocking existing product (Chips) --->>:
Product reloaded: Chips by 3 units.
Inventory after restocking Chips:
Chips - $1.50 (8 left)
Nutritional Info: 150 kcal, Stock: 8
Cola - $2.00 (3 left)
Nutritional Info: 120 kcal, Stock: 3
Candy - $1.00 (7 left)
Nutritional Info: 200 kcal, Stock: 7
Water - $1.20 (5 left)
Nutritional Info: 0 kcal, Stock: 5
```

The demo:

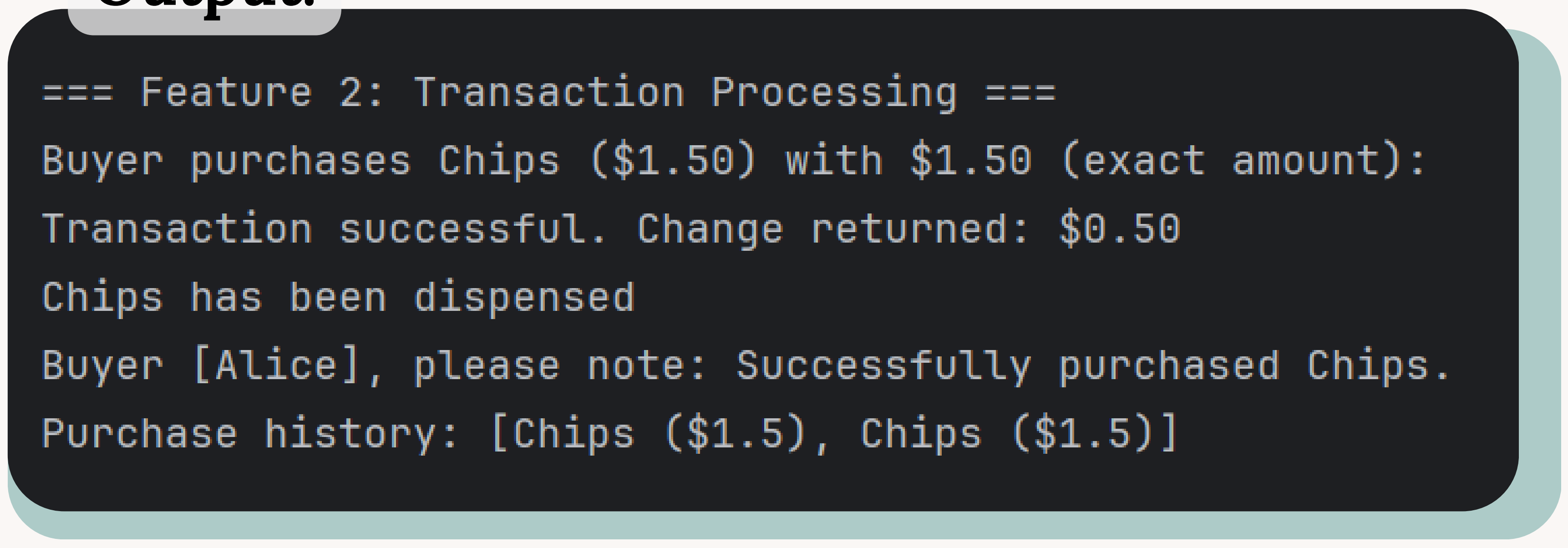
- Displays the initial inventory (Chips, Cola, Candy) with stock levels.
- Restocks an existing product (adds 3 Chips, increasing stock from 5 to 8).
- Adds a new product (Water with 5 units) to expand the inventory.
- Checks product availability (confirms Chips exists, Soda doesn't).
- Removes expired products (none expire, so inventory remains unchanged).

These actions showcase dynamic stock updates, product addition, availability queries, and expiry management, ensuring robust inventory control.

2. Transaction Processing:

This feature demonstrates the vending machine's transaction processing.

Output:



```
=== Feature 2: Transaction Processing ===
Buyer purchases Chips ($1.50) with $1.50 (exact amount):
Transaction successful. Change returned: $0.50
Chips has been dispensed
Buyer [Alice], please note: Successfully purchased Chips.
Purchase history: [Chips ($1.5), Chips ($1.5)]
```

The demo:

- Buyer (Alice) selects Chips ($1.50) and pays $2.00.
- Confirms item availability and sufficient payment.
- Adds Chips to buyer's purchase history.
- Dispenses Chips and returns change($0.50) resets session money.

These actions show secure payment handling and accurate purchase tracking.

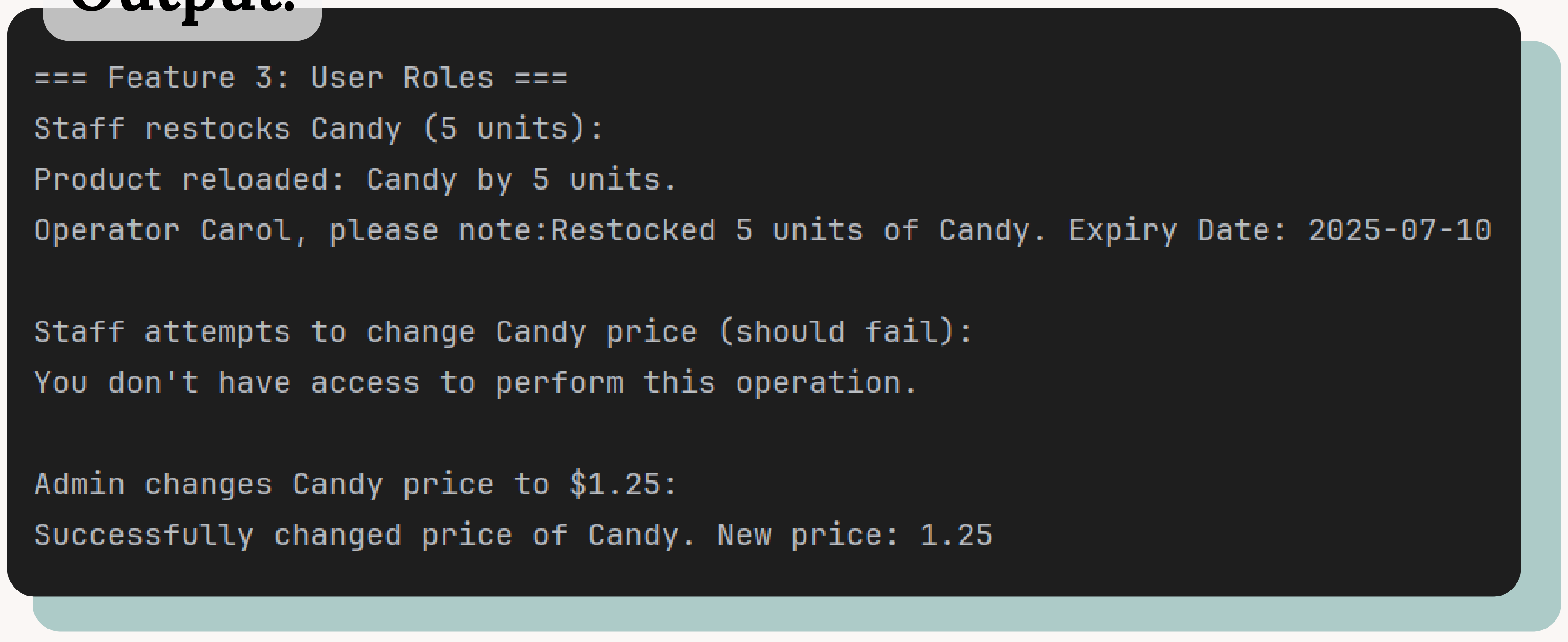
Program Features and Screenshots (Continued)

(All execution code available in Main class)

3. User Roles:

This feature showcases the vending machine's user role management.

Output:



```
=== Feature 3: User Roles ===
Staff restocks Candy (5 units):
Product reloaded: Candy by 5 units.
Operator Carol, please note:Restocked 5 units of Candy. Expiry Date: 2025-07-10

Staff attempts to change Candy price (should fail):
You don't have access to perform this operation.

Admin changes Candy price to $1.25:
Successfully changed price of Candy. New price: 1.25
```

These actions demonstrate role-based access control, ensuring only authorized users perform specific tasks.

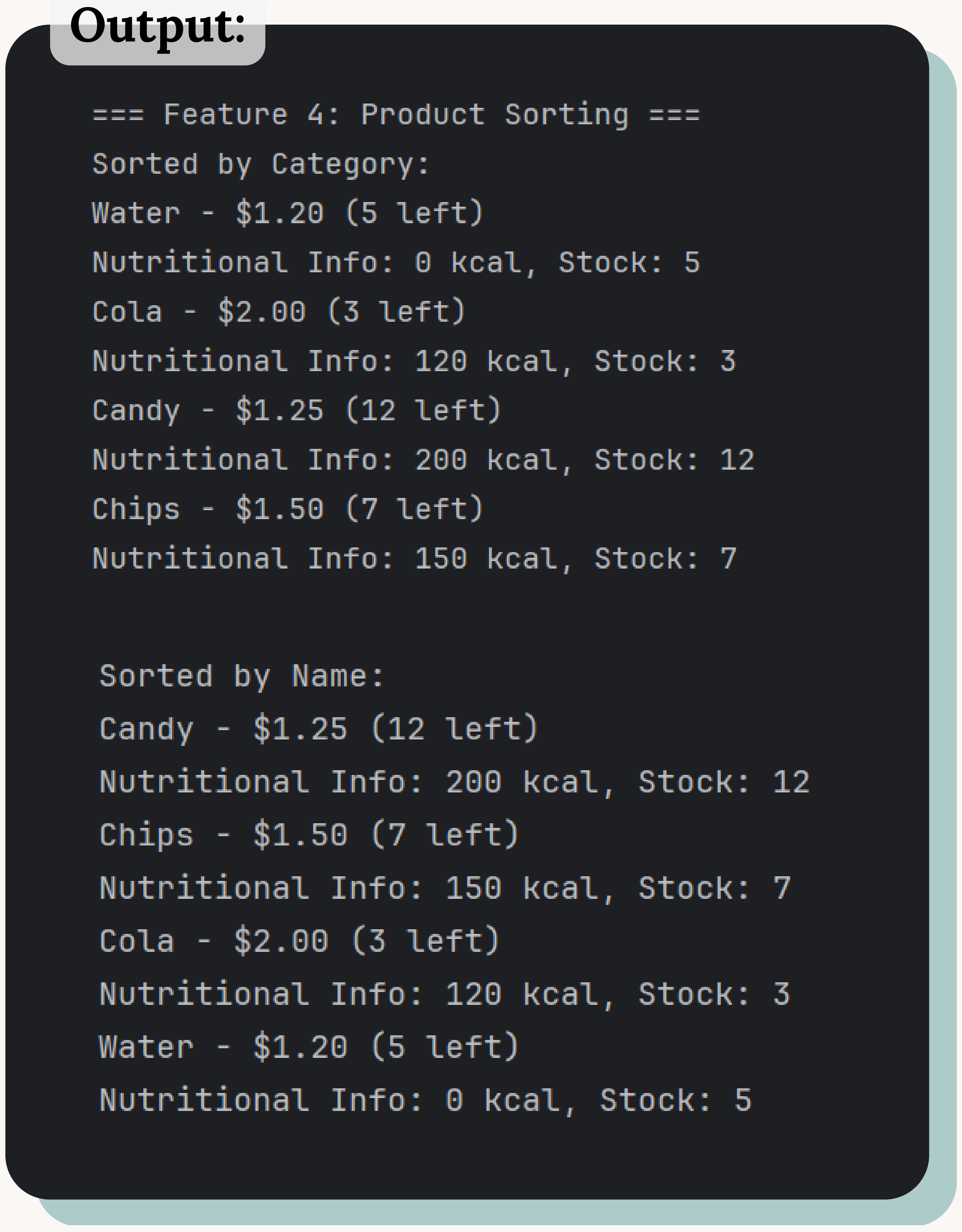
The demo:

- Staff (Bob) restocks Candy (5 units), updating inventory.
- Staff attempts to change Candy's price (fails due to lack of admin access).
- Admin (Carol) changes Candy's price to $1.25 successfully.

4. Product Sorting:

This feature demonstrates the vending machine's product sorting capabilities

Output:



```
=== Feature 4: Product Sorting ===
Sorted by Category:
Water - $1.20 (5 left)
Nutritional Info: 0 kcal, Stock: 5
Cola - $2.00 (3 left)
Nutritional Info: 120 kcal, Stock: 3
Candy - $1.25 (12 left)
Nutritional Info: 200 kcal, Stock: 12
Chips - $1.50 (7 left)
Nutritional Info: 150 kcal, Stock: 7

Sorted by Name:
Candy - $1.25 (12 left)
Nutritional Info: 200 kcal, Stock: 12
Chips - $1.50 (7 left)
Nutritional Info: 150 kcal, Stock: 7
Cola - $2.00 (3 left)
Nutritional Info: 120 kcal, Stock: 3
Water - $1.20 (5 left)
Nutritional Info: 0 kcal, Stock: 5
```

The demo:

- Sorts inventory by category (Drinks: Cola, Water; Snacks: Candy, Chips).
- Sorts inventory by name (alphabetically: Candy, Chips, Cola, Water).
- Displays sorted products with prices, stock, and nutritional info.

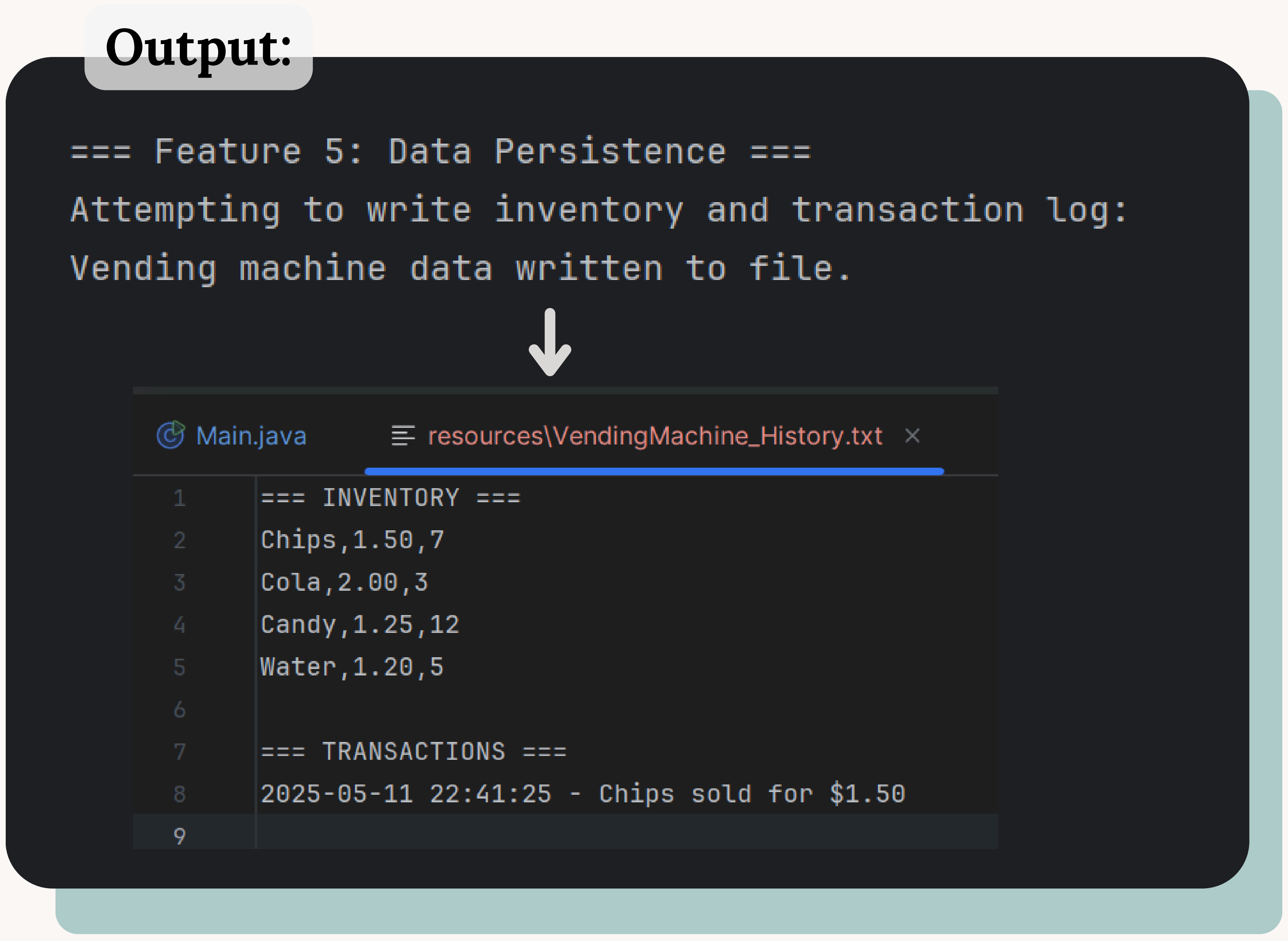
These actions showcase flexible inventory organization for user convenience.

Program Features and Screenshots (Continued)

(All execution code available in Main class)

5. Data Persistent

This feature demonstrates the vending machine's data storage capability.



Output:

```
=== Feature 5: Data Persistence ===
Attempting to write inventory and transaction log:
Vending machine data written to file.
```

Main.java | resources\VendingMachine_History.txt

```
=== INVENTORY ===
Chips,1.50,7
Cola,2.00,3
Candy,1.25,12
Water,1.20,5

=== TRANSACTIONS ===
2025-05-11 22:41:25 - Chips sold for $1.50
```

The demo:

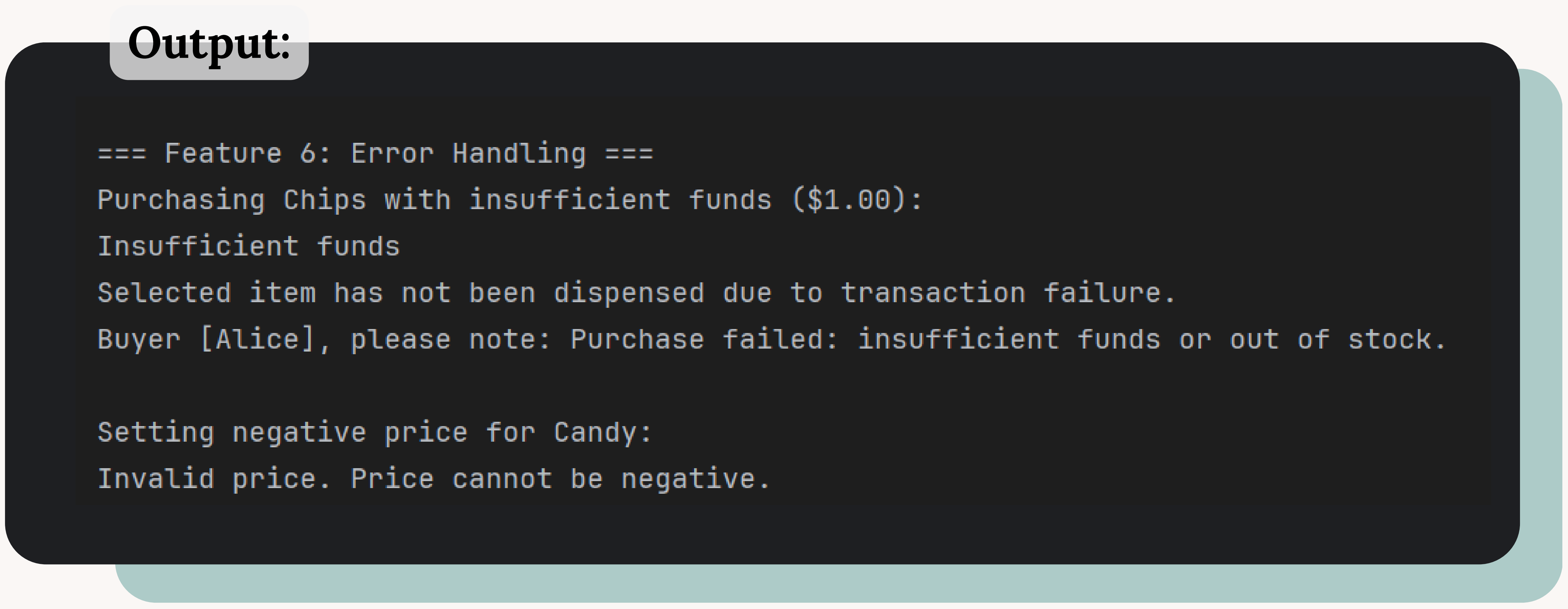
- Writes current inventory (product names, prices, stock) to a file.
- Logs transaction details (e.g., Chips sale) in the same file.

This action ensures inventory and sales data are saved for record-keeping.

---

6. Error Handling:

This feature demonstrates the vending machine's error handling.



Output:

```
=== Feature 6: Error Handling ===
Purchasing Chips with insufficient funds ($1.00):
Insufficient funds
Selected item has not been dispensed due to transaction failure.
Buyer [Alice], please note: Purchase failed: insufficient funds or out of stock.

Setting negative price for Candy:
Invalid price. Price cannot be negative.
```

The demo:

- Attempts to purchase Chips ($1.50) with insufficient funds ($1.00) -> failing.
- Attempts to set a negative price for Candy -> failing.

These actions show robust validation to prevent invalid transactions and pricing.

Project Development Challenges

While developing this project, I encountered a few challenges, mainly due to insufficient initial planning:

1. **Revisiting Method Parameters:** Since the design wasn't perfectly planned from the start, I noticed that several methods across different classes overlapped in functionality. This required me to go back and revise the parameters and purposes of some methods to avoid redundancy and ensure that each method had a clear, distinct role.

2. **Adjusting the Product Class:** I had to add and remove some methods from the *Product* class compared to what I originally planned. This was because, during implementation, I realized that certain operations—such as checking stock or modifying attributes—required different logic than I had initially expected.

3. **Connecting Classes:** One of the most challenging aspects was integrating and using methods across multiple classes. Learning how to pass objects correctly and remembering to call methods from one class within another took time and effort, especially when handling user interactions and product selection.

Overall, these challenges helped me better understand the importance of careful planning and solid class structure in an object-oriented program.

Learning Outcomes

Working on the vending machine project gave me hands-on experience with planning, coding, and managing a full development process. Here are some key takeaways:

Planning and Design

- Thorough Class Planning: I learned that planning goes beyond listing features—it's about clearly defining each class's role and how they interact. Since my initial design lacked a well-defined structure, I ended up having to tweak method logic and class relationships.

- Defining a Core Scope: Balancing ambitious and creative ideas with time constraints was challenging. Defining a clear set of essential features from the start helped me stay focused and reduce the need for last-minute fixes.

Code Quality

- Writing Clear, Understandable Code: Using descriptive method names and basic documentation for each method's purpose, inputs, and outputs made a big difference when connecting classes. It also made the code much easier to revisit and debug.

- Breaking Down and Structuring Code: Refactoring taught me the value of keeping methods short and focused on a single task. Implementing interfaces where appropriate made the system more modular and easier to maintain or expand in the case needed.

Development Practices

- Testing Early and Often: Although I did follow test-driven development, I realized that even writing simple tests during development helped me catch issues like overlapping methods and logic errors much earlier.

- Using Version Control Effectively: Making smaller, well-labeled commits in Git made it much easier to track progress and undo mistakes.

Overall, this project showed me that thoughtful planning, clean code, and good development habits go a long way in creating a smooth workflow and a solid final product.